Architecture Evaluation Product Documentation

Mikael Lindvall, Fraunhofer Center

As software systems become increasingly complex, the need to investigate and evaluate them at high levels of abstraction becomes more important. When systems are very complex, evaluating the system from an architectural level is necessary in order to understand the structure and interrelationships among the components of the system.

The architectural evaluation technology is a process that assures that the actual implementation of a software architecture in source code matches specified architectural guidelines to produce and maintain quality software. Accompanying software tools support this process.

This seven-step process was designed to be relatively quick and used repeatedly to keep the actual implementation of the software architecture consistent with the planned architectural design. It does not assume that documentation of the architectural design exists prior to the evaluation of the system. However, if documentation does not exist prior to the evaluation, the process requires input from the developers of the system to recover the planned architectural design of the system. The architectural evaluation process involves members of the development team and ideally, a separate analysis team. One of the main objectives of this process is to provide useful feedback to the development team without extensively disrupting their daily activities. The members of the development team are used sparingly in this process to confirm the findings of the analysis team.

Step 1. Select a perspective for evaluation

Architectural evaluations may be conducted with different goals in mind and from many different perspectives. For example, a system might be evaluated to determine whether or not the system implements the specified functional requirements or to determine if it fulfills the non-functional requirements, i.e. the system qualities or quality attributes. Other examples of perspectives are evaluation for maintainability, reusability, flexibility or evolvability, security, reliability, and performance. Selecting a perspective is important for identifying appropriate goals and measurements for the evaluation process. The GQM[1] technique is used to define goal-oriented metrics based on questions that need to be answered to determine if goals have been achieved. For example, if the perspective chosen is maintainability, goals based on attributes of maintainability such as coupling and cohesion might be identified.

The analysis team performs this step with the help of the development team. The development team provides input on the perspective. The analysis team creates the goals and defines the metrics following the GQM technique. The development team provides feedback to the analysis team on the goals and metrics.

Step 2. Define planned architecture and guidelines

Once a perspective has been chosen and goals have been identified and elaborated with the GQM technique, the planned architecture of the system is identified and guidelines with associated metrics are defined. These architectural guidelines are used to validate that the architecture possesses desired properties. Often, these guidelines translate into quantitative metrics. For example, if evaluating a system from the perspective of maintainability, guidelines related to coupling might be established. Sample guidelines based on coupling might include the following: coupling between the components should be low; and the extent of the coupling between components should be low. Quantitative metrics measuring coupling between the components and the extent of the coupling are derived from these guidelines. In addition to defining guidelines and metrics based on the perspective of the architectural evaluation, some guidelines and metrics are defined based on the architectural styles and the design patterns chosen for the system.

The analysis team works with one or two representatives of the development team (and/or uses documentation) to identify the planned, high-level architecture of the system. The planned (or ideal/intended) architecture is defined by architectural requirements, by implicit and explicit architectural

---

[1] Basili V. R., Caldiera G., and Rombach D. H., The Goal Question Metric approach, Encyclopedia of Software Engineering - 2 Volume Set Wiley, pp. 528-532, 1994.

guidelines and design rules and implications stemming from the use of architectural styles and design patterns. The analysis team needs to recover the different aspects of the planned architecture and create a model of it that will guide the evaluation. Once the high level architecture of the system has been defined, the analysis team uses it to derive the implications, trade-offs, guidelines, and design rules that result. The analysis team needs to select and customize the guidelines and metrics for the specific context. The selected set of metrics must capture the properties that the team finds most important while, at the same time, being cost-efficient to collect and analyze. As the analysis team learns more about the planned architecture, these guidelines and metrics are commonly iterated and updated during this step.

Step 3. Recover actual architecture

The actual architecture is the high-level structure of the implemented system, its architectural components and their interrelationships, as well as its architectural styles and design patterns. Studying the implementation of the system, which is, to a large extent an abstraction obtained from the source code, represents the actual architecture. It should be noted that this step is not the same as source code analysis, but it is used to identify the static architectural components of the actual system. To perform this step efficiently, the analysis team relies on a set of automated or partially automated tools that help them with this task. In many cases, the tools have to be defined based on programming language, the measurements that are to be collected, and other factors in the development environment.

Identifying the contents of a component is often one of the key complications involved with the recovery of the high-level architecture of an implemented system. In some cases, programming language features can be used to reduce some of the difficulties associated with this task. With Java, for example, the analysis team can use packages as a way of determining the contents of the system's components. However, not all Java developers use packages and even when packages are used, there is not always a one-to-one correspondence between the packages and the high-level components of the system.

Identifying architectural styles and design patterns is another complication that arises with the recovery of the actual architecture. Architectural styles are not always easy to detect in the actual implementation of a system. Design patterns can be implemented in different ways and can be difficult to detect.

As part of this step of the process, the design team works with one or two members of the development team to partition the files containing the actual implementation of the system into their appropriate components. Then, the analysis team extracts relevant information and computes metrics from the component files to obtain the actual architecture of the system.

Step 4. Compare actual to planned to identify architectural deviations

Architectural deviations are differences between the planned architecture and the actual implemented version of the architecture. These architectural violations are identified by comparing the planned architectural design defined in step two to the abstraction of the actual architecture obtained in step three. Deviations can be missing or extra components, missing or extra connections between components, violations of architectural guidelines, or values of metrics that exceed or do not match a certain expected value.

The analysis team compiles a list of these violations and notes the circumstances under which the violation was detected and the reason the team suspects it is a violation. If necessary, the analysis team conducts a more detailed analysis of the deviation in order to determine its possible cause and degree of severity. The deviations are categorized and patterns of violations are identified.

Step 5. Verify violations

Once the analysis team has composed and characterized the list of architectural violations, the list is verified during a discussion with one or two members of the development team. This step is taken for several reasons. First, it helps to ensure that the analysis team has not incorrectly identified any violations by a misunderstanding of how the system was implemented. Secondly, it gives the development team feedback on how well the actual implementation matches the planned architecture and exposes the general types of deviations that have occurred. Additionally, the analysis team gathers more information on how and why the violations have occurred.

Step 6. Suggest changes to planned and/or actual architectures

Based on the results from the previous step, the analysis team formulates high-level change recommendations that would remove the deviations from the system. Sometimes, the deviations result in requests for source code changes. In some cases, the requests are related to changes in the planned architecture or guidelines. It should be noted that it is not the task of the analysis team to design or implement the change requests. Rather, this step is a way for the analysis team to contribute to the improvement of the system in a constructive way by giving feedback to the development team.

Step 7. Repeat steps 4 – 6 after changes have been implemented

The analysis team discusses the change requests with the development team. It is the role of the development team to decide which changes to implement and how the changes will be implemented. Once the changes have been implemented, it is important to verify that actual architecture complies with the planned one. To verify that the planned and actual architectures are in alignment, the steps of identifying the actual architecture and any architectural deviations are repeated. This verification is done to make sure that the changes have been implemented correctly and that no new violations have been introduced into the system. The steps of the process are shown in Figure 1.

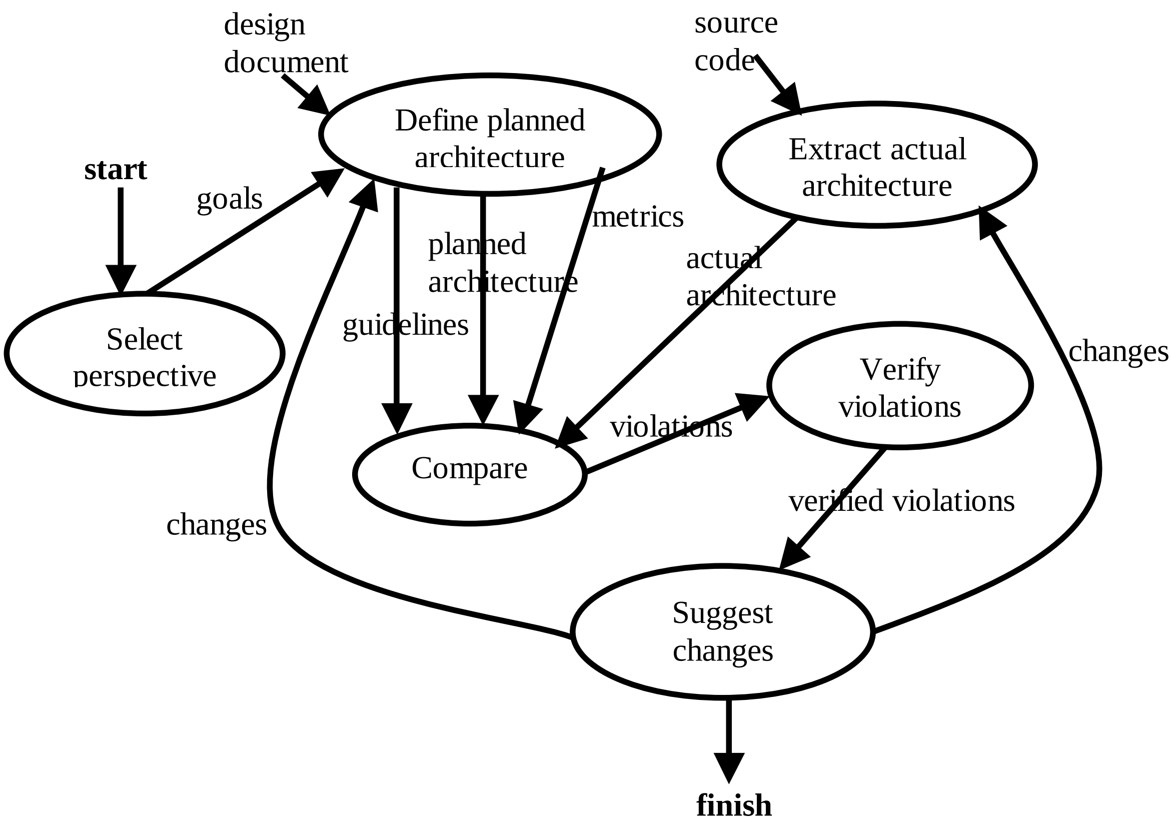

Figure 1: A diagram of the software architectural evaluation process steps.

Use of the process

The process for software architectural evaluation is designed to be efficient and conducted at various points over the lifetime of a software system. It is expected that one iteration of the evaluation process will be run after a new feature or set of features is implemented. Another important objective in the design of this process was to provide valuable feedback to the development team with minimal disruption to the team's usual activities. The process was designed to fit seamlessly into an existing development environment. The analysis team should perform the bulk of the work needed to conduct the evaluation. Members of the development team should be used to verify the findings of the analysis team in both the definition and recovery of the architecture and implement the recommended changes.